Scaling the Java™ Technology Environment in Four Dimensions

Jim Waldo

Distinguished Engineer
Sun Microsystems Laboratories
http://research.sun.com

Session TS-3625

Goal of This Talk

Learn how to identify and refer to services that may change their location over time, as a first step to building long-lived systems

Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A UuidDirectory

Refreshable References

In Conclusion

Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A UuidDirectory

Refreshable References

In Conclusion

The Driving Application

Medical sensing

- Suppose you could
  - Place medical sensors on everyone
  - Record the information all the time
- We have the sensor technology
- We have the network infrastructure
- We don't have the back-end technology

A Multi-Purpose Infrastructure

- Practicing physicians
  - Allow better diagnosis
  - Enable triage at a distance
- Public health
  - Real-time monitoring
- Research
  - A rich data set for longitudinal studies

Such a System Would Require

- A distributed solution
  - Nothing else could scale
  - Nothing else would give appropriate security
- A reliable solution
- A federated solution
  - There is not central medical authority
- A long-lived solution
  - Data needs to last as long as the patients

Building an 80 Year System

- Every part of the system will be replaced
  - Often more than once
- Services will evolve
- Information will change
- Services will move
  - To balance load
  - As machines are replaced

Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A UuidDirectory

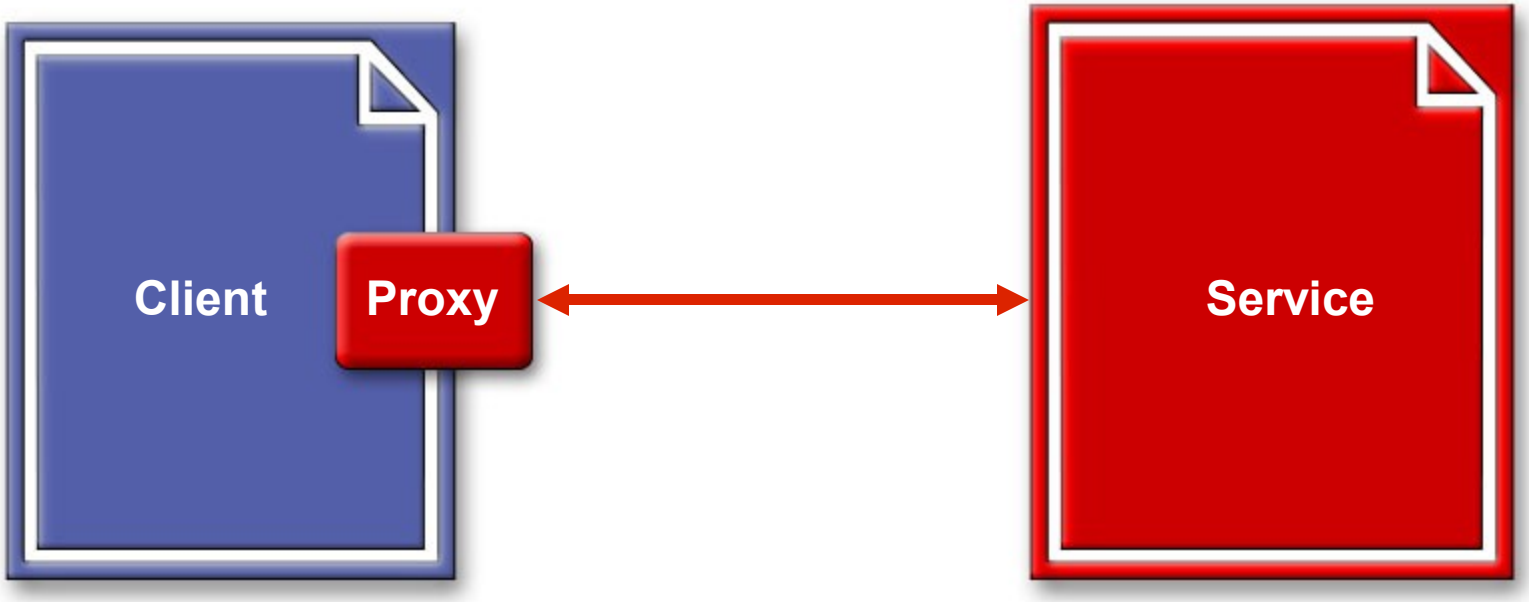
Refreshable References

In Conclusion

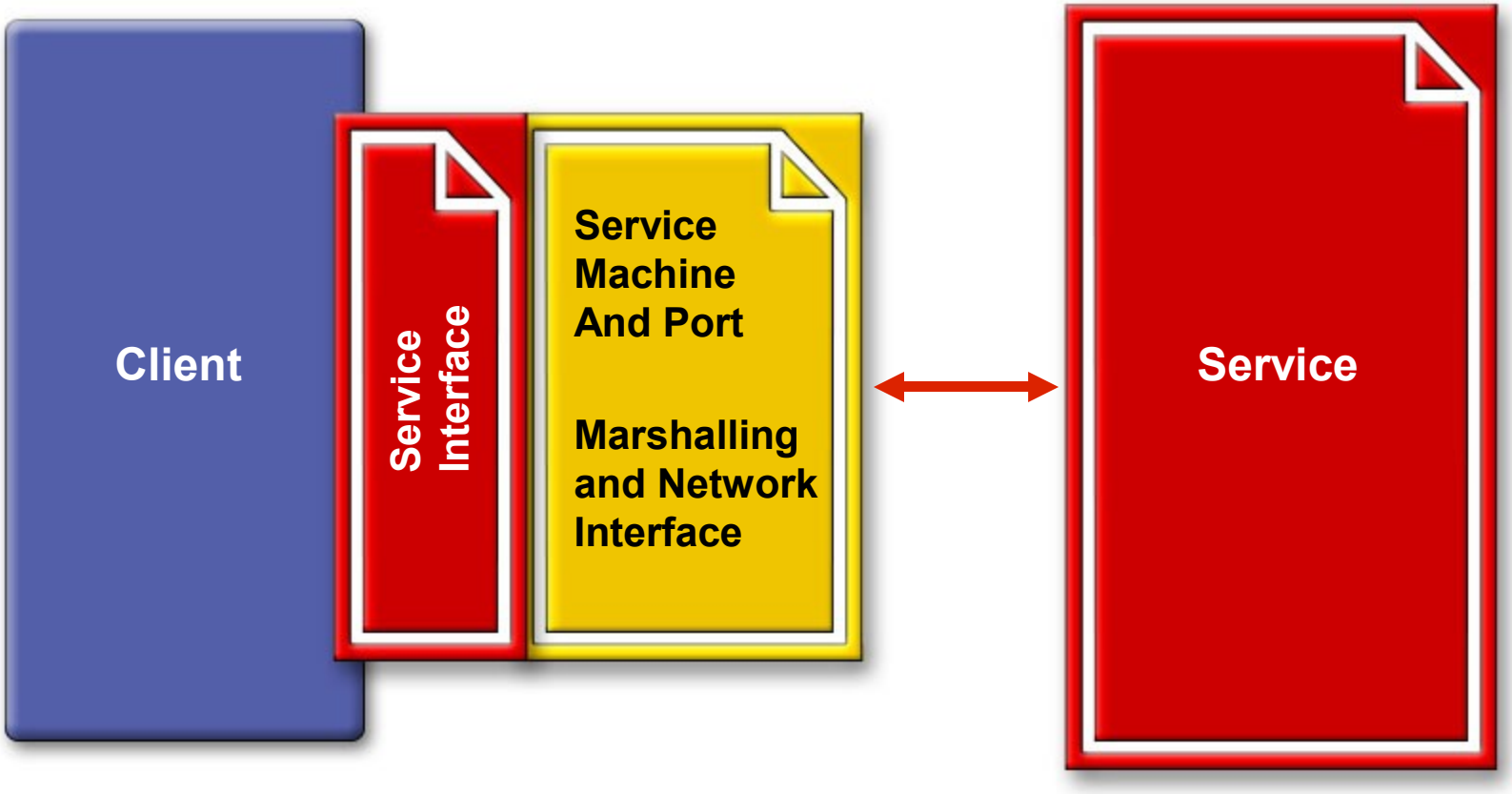
A Remote Reference

- Supports a service interface
- Contains location information
  - The machine where the server is
  - The port where the server is listening
  - The machine and port that knows where the server is
- This is true of
  - Java Remote Method Invocation (RMI)
  - CORBA
  - Jini™ Extensible Remote Invocation

The Big Picture



Somewhat More Detail



When a Service Moves

- The reference points to the wrong machine
- We could leave a tombstone
  - Tombstones forward from old location to new
  - But what if the machine is gone?
- We could require all references to be replaced
  - Inefficient
  - How do we insure we get the same service?

What About Using Names?

All problems can be solved with an extra indirection

- Identify a service by a name
  - Provide a naming service
  - When a service moves, get a new proxy by name
- Usual names are hierarchical
  - Local scope allows guaranteed uniqueness
  - Hierarchy enables scale
- But hierarchies introduce location
  - Movement assumes no location

Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A UuidDirectory

Refreshable References

In Conclusion

A Different Kind of Naming

Pure names

- A pure name can only be used
  - To compare for equality
  - To identify an object
- Also known as Universally Unique Identifiers
- There are multiple instantiations
  - **`java.util.UUID`**
  - IETF draft
  - Home-grown (patient identifiers)
- How to federate?

A Different Kind of Naming Service

How can we scale a flat namespace?

- Usually name services scale with
  - Local name service
  - Hierarchy based on name
  - Traverse lookup hierarchy from name hierarchy
- Pure names have no hierarchy
- Use distributed hash tables
  - Partition name space arbitrarily
  - Connect naming services

Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A UuidDirectory

Refreshable References

In Conclusion

Pure Name Requirements

One approach does not fit all

- Identifiers must be
  - Unique (to what level of confidence?)
  - Comparable
  - Allow mapping to existing identifiers
- The base intuition
  - Make part of the identifier how it was generated
  - Allow (but not require) global disambiguation

Xuid Interface

```
Interface Xuid {
    public long getGeneratorId();
    public byte[] getId();
}

// any class implementing this will need to implement
// equals() and hashcode() so that
// a.equals(b) iff a and b have the same GeneratorId and
// the byte array of the rest of the Xuid is byte-for-byte
// the same
```

Xuid Layout

A two-part universally unique identifier, first part

- The GeneratorId
  - If registered with a global authority
    - First two bits are 0
    - Next 62 bits are guaranteed unique by the authority
  - Otherwise
    - First 64 bits are a secure hash of the generation code
- There may not be a central authority
- Two unregistered generators will be equal
  - Only if the generation code is identical
  - Which is what we want

Xuid Layout

A two-part universally unique identifier, second part

- The Id
  - A byte array
    - Different generators may produce different sizes
    - Generators are responsible for uniqueness
- Identity comparisons
  - Should check generator Id first
  - If those are the same, byte length should be the same
- Hashing
  - Combine generator and bytes

Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A XuidDirectory

Refreshable References

In Conclusion

XuidDirectory Interface

```
Interface UuidDirectory
{
    Lease register(Xuid id, Object o, long leaselen)
                throws UnknownXuidException,
                       RemoteException;

    Lease[] register (Xuid[] ids, Object[] o,
                        long[] leaseLens)
                throws UnknownXuidException,
                       RemoteException;

    Object lookup(Xuid id, XuidDirectory[] visited)
                throws UnknownXuidException,
                       RemoteException;
}
```

Xuid Directories

- Registration
  - Of single objects
  - Of groups of objects
  - Objects responsible for their own registration
- All registrations are leased
  - Bounded time failure determination
- `XuidDirectory` objects can
  - Consult other directories
  - Split and join
  - `Lookup` needs to know where it has been

Xuid Directories as DHTs

- Distributed Hash Tables
  - Partition the space of a hash
  - Dynamically split and join
  - Have failure recovery mechanisms
- A good fit for Xuid Directories
  - Flat namespace
  - Excellent scaling properties
- Peer-to-peer properties
  - Allow merging disjoint directories on discovery
  - A first step at federation

Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A XuidDirectory

Refreshable References

In Conclusion

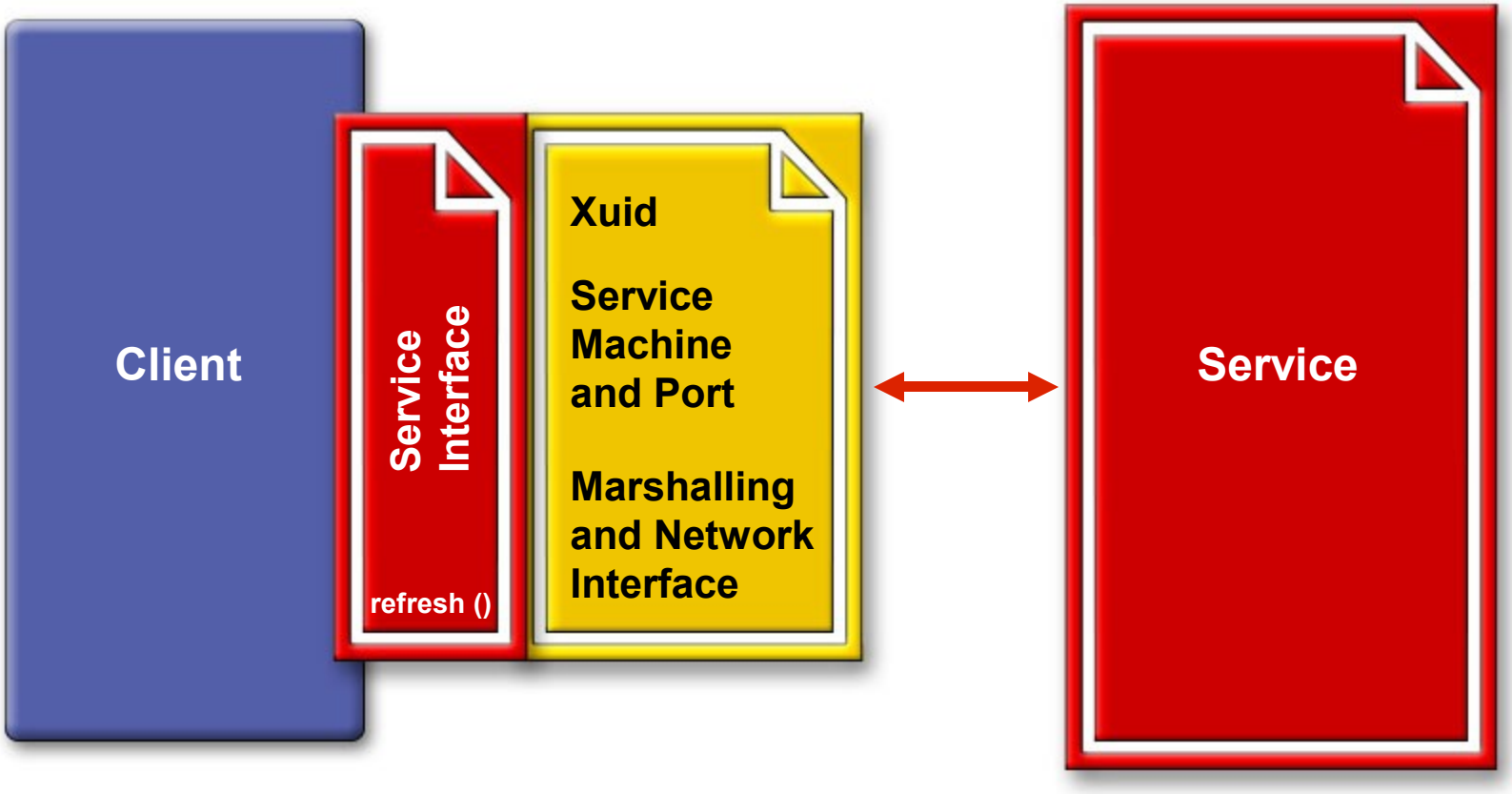
Refreshable References

```
Interface BasicRef
{
    Xuid getXuid();
    Object refreshProxy(XuidDirectory[] hints)
              throws UnknownXuidException,
                     RemoteException;
}
```

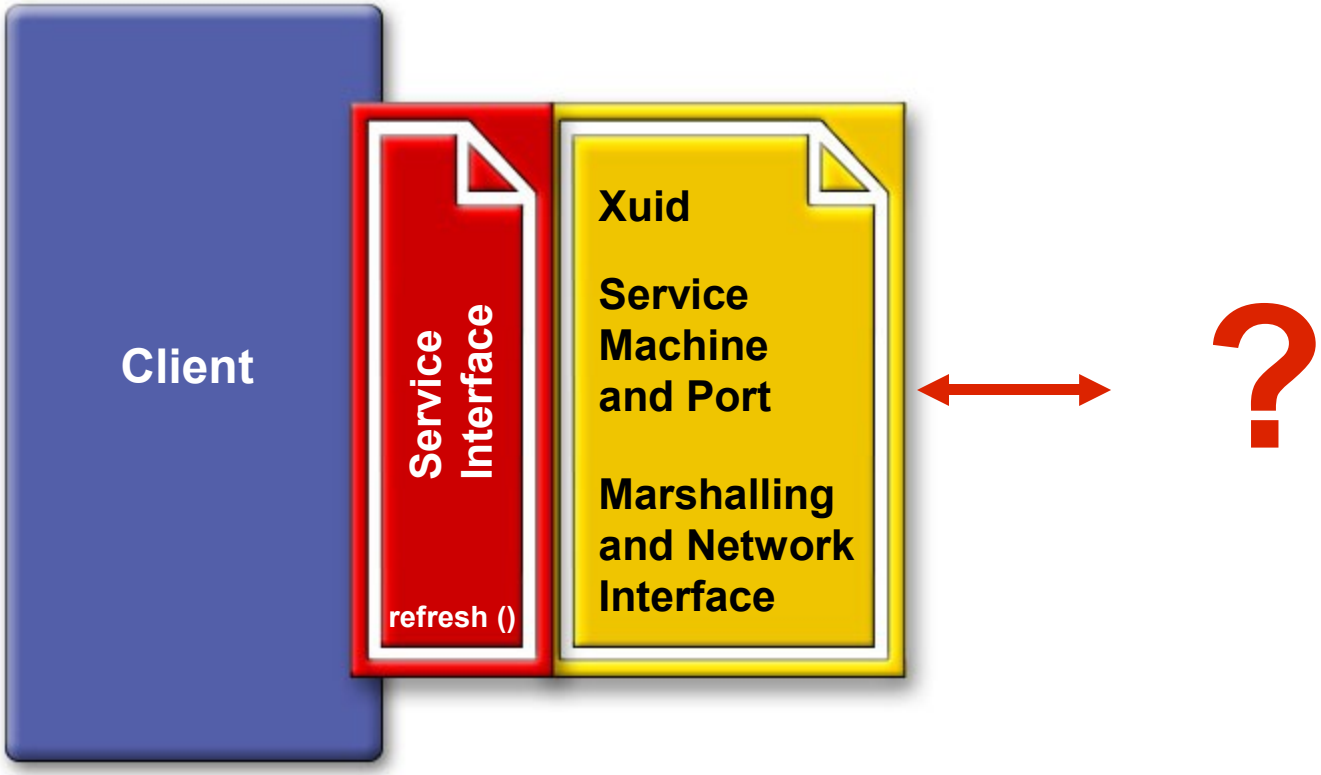
References to Moveable Objects

- Implement BasicRef
  - Need an Xuid
  - Implement a refresh method
- On receiving an exception, the client
  - Calls refresh()
  - Can supply local directories
  - Casts the result to the right type
  - Discards the old reference
  - Can continue

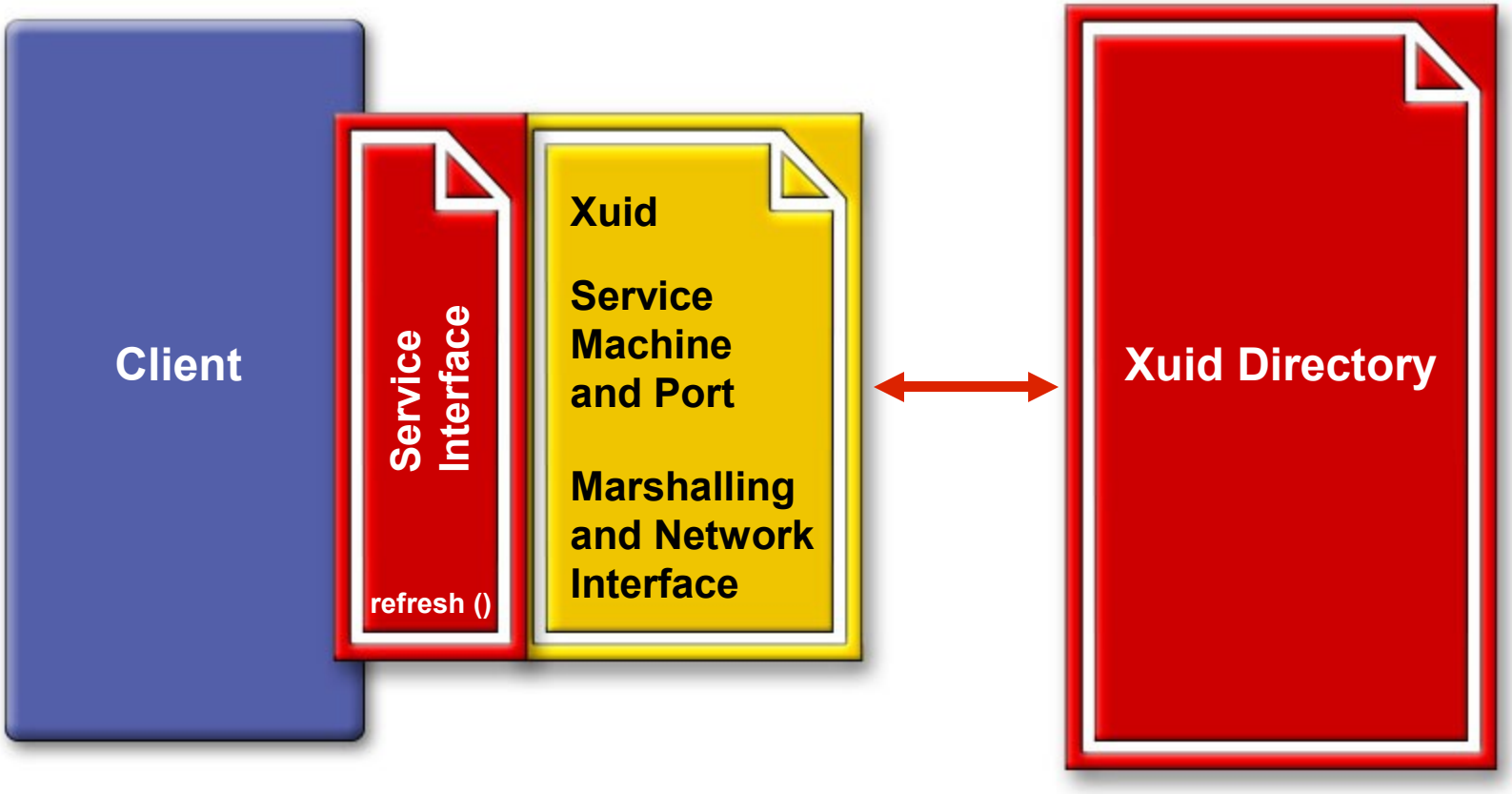
Somewhat More Detail (Revisited)



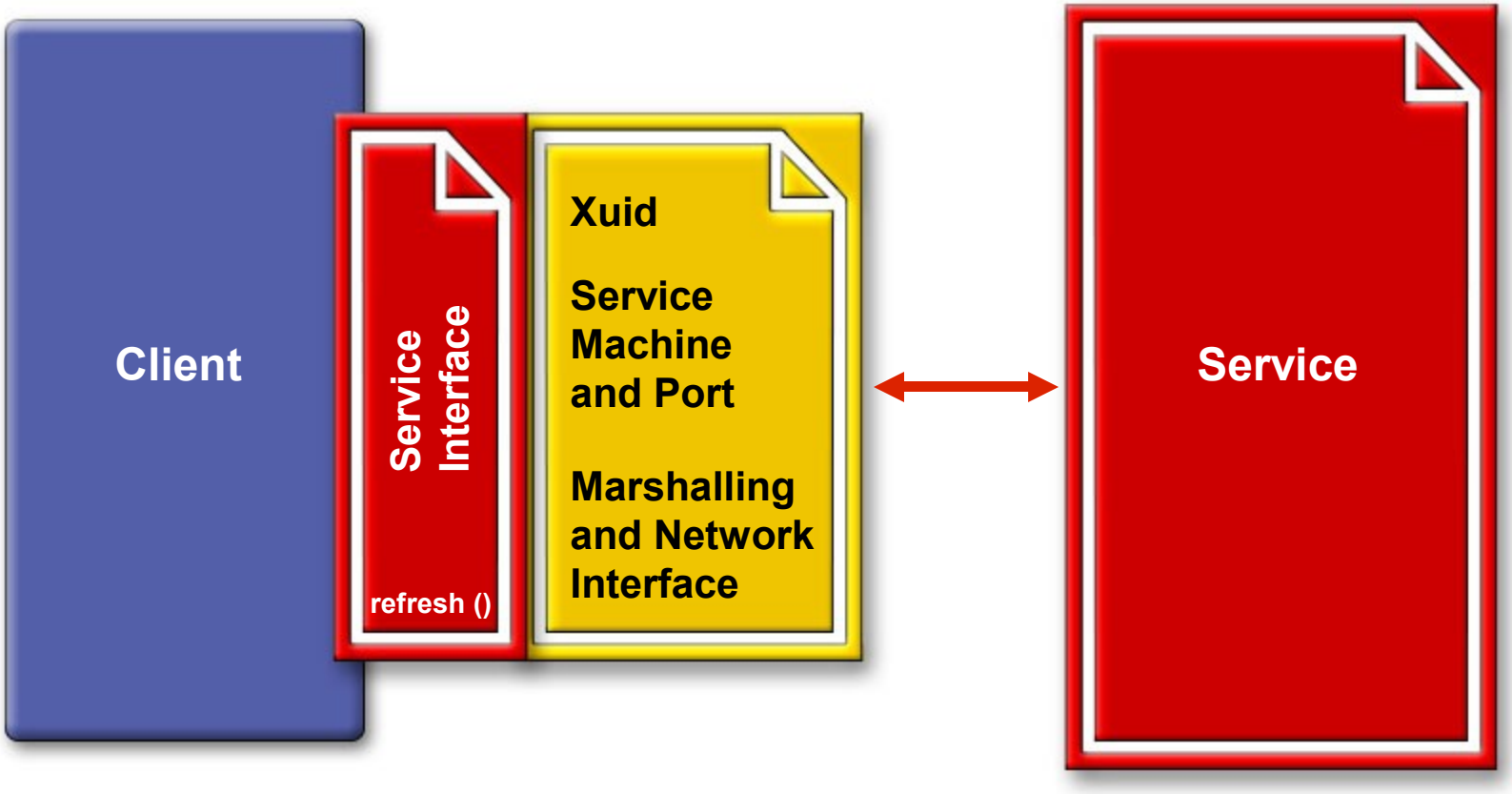
When the Service Moves



Use the Xuid Directory



Use the New Reference



Agenda

Setting the Stage

Current Remote References

Sketch of the Solution

Extensible Pure Names

A UuidDirectory

Refreshable References

In Conclusion

Services Can Move

But still might get lost

- Xuids give
  - Pure names with no presumed location
  - Uniqueness tied to the way they are generated
- Xuid directories allow
  - Moved objects to re-register
  - Refreshable proxies to find the new location
- But things can still fail
  - Service might just be gone
  - May have moved to an inaccessible Xuid Directory

Just a First Step

There is lots more to do

- Security model
  - Need to insure security and privacy
  - But medicine has a different model
- More than location will change
  - There will be new sensor/data types
  - There will be new languages
  - There will be new things that we can't conceive
- But this is research
  - Which entails that we don't know what we are doing

Q&A

Jim Waldo

Scaling the Java™ Technology Environment in Four Dimensions

Jim Waldo

Distinguished Engineer
Sun Microsystems Laboratories
http://research.sun.com

Session TS-3625